WORKFORCE SCHEDULING

Outline

- ❑ Days-Off Scheduling
- ❑ Shift Scheduling
- ❑ Cyclic Staffing Problem
- ❑ Applications and Extensions of Cyclic Staffing
- ❑ Crew Scheduling
- ❑ Operator Scheduling in a Call Center

Workforce timetabling

- ❑ Arrange shifts and assign people to them
- ❑ **Constraints**:
  - Number of people per shift
  - Minimum days off (*x*/*k* days must be off)
  - Weekends
- ❑ Nurses, call centers, hotels, restaurants, plane crew, factories, etc.

Days-Off Scheduling

- ❑ Days-Off Scheduling

NOT

Off-Days Scheduling:

"Scheduling workers who fall asleep on the job is not easy."

Days-Off Scheduling

- ❑ Number of workers assigned to each day
- ❑ Fixed size of workforce

➢ **Problem**: find minimum number of employees to cover a week operation

Constraints

- ❑ Demand per day n_j, $j = 1,2,\ldots,7$ (n_1 is Sunday; n_7 is Saturday)
- ❑ Each employee is given k_1 out of every k_2 weekends (days 1 and 7) off
- ❑ Each employee works 5 out of 7 days
- ❑ Each employee works no more than 6 consecutive days

Optimal schedule

➢ **Objective**: find minimum workforce size W

❑ Optimal schedule is generated for one week at a time
- schedule for week i+1 is determined after schedule of week i, and so on.

❑ Cyclic optimal schedule exists that it repeats itself.

Lower bounds on minimum size of *W*

❑ Weekend constraint

$$(k_2 - k_1)W \geq k_2 \max(n_1, n_7) \;\rightarrow\; W \geq \left[\frac{k_2 \max(n_1, n_7)}{k_2 - k_1}\right] = B_1$$

❑ Total demand constraint

$$5W \geq \sum_{j=1}^{7} n_j \quad \text{or} \quad W \geq \frac{1}{5}\sum_{j=1}^{7} n_j = B_2$$

❑ Maximum daily demand constraint

$$W \geq \max(n_1, n_2, \ldots, n_7) = B_3$$

Optimal schedule

$$W = \max(B_1, B_2, B_3)$$
$$n = \max(n_1, n_7)$$
$$u_j = \begin{cases} W - n_j & j = 2,\ldots,6 \\ n - n_j & j = 1,7 \end{cases}$$

❑ u_j is the surplus number of employees in day j

❑ The second lower bound guarantees:

$$\sum_{j=1}^{7} u_j \geq 2n$$

Algorithm

❑ Schedule weekends off

❑ Determine additional off days (in pairs)

❑ Categorize employees

❑ Assign off-day pairs to employees

➢ Can be shown that schedule is feasible and optimal

➢ A cyclic optimal schedule exists

Example 12.2.2 (Pinedo)

❑ Consider the following requirements:

day	1	2	3	4	5	6	7
	Sun	Mon	Tue	Wed	Thu	Fri	Sat
Requir.	1	0	3	3	3	3	2

❑ Employee requires 1 out of 3 weekends off: k_1=1, k_2=3.

➢ $n = \max(n_1, n_7) = 2$

Step 1: Weekends off

❑ Compute the minimum workforce:

$$W \geq [(3\times 2)/(3-1)] = 3,$$
$$W \geq [15/5] = 3,$$
$$W \geq 3$$

❑ Thus, $W = 3$ and $W - n = 1$.

❑ Assign 1st **weekend off** to first $W - n$ employees, 2nd weekend off to the next $W - n$ and so on.
- Employee 1 comes after employee W
- Cycle until you assign employee 1 to the same weekend off again

Step 1: Weekends off

day	1	2	3	4	5	6	7
	Sun	Mon	Tue	Wed	Thu	Fri	Sat
Req.	1	0	3	3	3	3	2

$W = 3$
$n = 2$
So, 1 employee is off each weekend

	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S
1																						
2																						
3																						

Step 1: Weekends off

day	1	2	3	4	5	6	7
	Sun	Mon	Tue	Wed	Thu	Fri	Sat
Req.	1	0	3	3	3	3	2

$W = 3$
$n = 2$
So, 1 employee is off each weekend

	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S
1	X	X																				X
2								X	X													
3															X	X						

Step 2: Construct off-day pairs

- There are 1 surplus employee on Sunday and 3 on Monday:

day	1	2	3	4	5	6	7
	Sun	Mon	Tue	Wed	Thu	Fri	Sat
u_j	1	3	0	0	0	0	0

- Construct a list of day-pairs with over-capacity

Step 2: Construct off-day pairs

- Pick u_k
- Pick another day u_m ($m \neq k$), $u_m > 0$
  - If all $u_m = 0$, $m \neq k$, choose $m = k$
- Add (k, m) to list and decrease u_k and u_m by 1
- Repeat n times
- Pairs (k, k) are called "non-distinct" pairs

Step 2: Construct off-day pairs

$W = 3$
$n = 2$

day	1	2	3	4	5	6	7
	Sun	Mon	Tue	Wed	Thu	Fri	Sat
Req.	1	0	3	3	3	3	2
u_j	1	3	0	0	0	0	0

(Mon, Sun)

$u_k = u_2 = 3$
$u_m = u_1 = 1$

Step 2: Construct off-day pairs

$W = 3$
$n = 2$

day	1	2	3	4	5	6	7
	Sun	Mon	Tue	Wed	Thu	Fri	Sat
Req.	1	0	3	3	3	3	2
u_j	0	2	0	0	0	0	0

(Mon, Sun)
(Mon, Mon)

$u_k = u_2 = 2$
$u_m = u_1 = 2$

Step 2: Construct off-day pairs

$W = 3$
$n = 2$

day	1	2	3	4	5	6	7
	Sun	Mon	Tue	Wed	Thu	Fri	Sat
Req.	1	0	3	3	3	3	2
u_j	0	0	0	0	0	0	0

(Mon, Sun)
(Mon, Mon)

- The pairs are: (Mon, Sun) and (Mon, Mon)

Step 3: Categorize workers

- Week 1 employees fall into 4 groups:
  - Type 1: weekend 1 off, 0 off days, weekend 2 off
  - Type 2: weekend 1 off, 1 off day, weekend 2 on
  - Type 3: weekend 1 on, 1 off day, weekend 2 off
  - Type 4: weekend 1 on, 2 off days, weekend 2 on
- n people working each weekend, so
  - $|T3|+|T4| = n$ (weekend 1)
  - $|T2|+|T4| = n$ (weekend 2)

Thus $|T3| = |T2|$

Step 3: Categorize workers

$W = 3$ (Mon, Sun)
$n = 2$ (Mon, Mon)

day	1	2	3	4	5	6	7
	Sun	Mon	Tue	Wed	Thu	Fri	Sat
Req.	1	0	3	3	3	3	2

T1 = { }
T2 = {1}
T3 = {2}
T4 = {3}

	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S
1	X	X																				X
2								X	X													
3															X	X						

Step 3: Categorize workers

- Assign off-day pairs to workers
- First to T4 – they get both days off
- Second to (T2,T3) pairs
  - T3 gets earlier day
  - T2 gets later day

Step 4: Assign off-day pairs

$W = 3$ (Mon, Sun)
$n = 2$ (Mon, Mon)

day	1	2	3	4	5	6	7
	Sun	Mon	Tue	Wed	Thu	Fri	Sat
Req.	1	0	3	3	3	3	2

T1 = { }
T2 = {1}
T3 = {2}
T4 = {3}

	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S
1	X	X	X																			X
2			X					X	X													
3		X	X												X	X						

Step 4: Assign off-day pairs for week *i*

- Categorize workers for week i
- Case 1: All off-days are distinct
  - $T4(i) = T4(i-1)$, $T3(i) = T3(i-1)$
  - T4 gets both days off, T3 gets earlier day, T2 gets later day
- Case 2: Not all off-days are distinct
  - Week i schedule is identical to week 1

Step 4: Assign off-day pairs for week 2

$W = 3$ (Mon, Sun)
$n = 2$ (Mon, Mon)

day	1	2	3	4	5	6	7
	Sun	Mon	Tue	Wed	Thu	Fri	Sat
Req.	1	0	3	3	3	3	2

T1 = { }
T2 = {2}
T3 = {3}
T4 = {1}

	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S
1	X	X	X																			X
2			X					X	X													
3		X	X												X	X						

Step 4: Assign off-day pairs for week 2

$W = 3$ (Mon, Sun)
$n = 2$ (Mon, Mon)

day	1	2	3	4	5	6	7
	Sun	Mon	Tue	Wed	Thu	Fri	Sat
Req.	1	0	3	3	3	3	2

T1 = { }
T2 = {2}
T3 = {3}
T4 = {1}

	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S
1	X	X	X						X	X												X
2			X					X	X	X												
3		X	X							X					X	X						

Step 4: Assign off-day pairs for week 3

$W = 3$ (Mon, Sun)
$n = 2$ (Mon, Mon)

day	1	2	3	4	5	6	7
	Sun	Mon	Tue	Wed	Thu	Fri	Sat
Req.	1	0	3	3	3	3	2

T1 = { }
T2 = {3}
T3 = {1}
T4 = {2}

	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S
1	X	X	X						X	X												X
2			X					X	X	X												
3		X	X							X					X	X						

Step 4: Assign off-day pairs for week 3

$W = 3$ (Mon, Sun)
$n = 2$ (Mon, Mon)

day	1	2	3	4	5	6	7
	Sun	Mon	Tue	Wed	Thu	Fri	Sat
Req.	1	0	3	3	3	3	2

T1 = { }
T2 = {3}
T3 = {1}
T4 = {2}

	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S
1	X	X	X						X	X							X					X
2			X					X	X	X						X	X					
3		X	X							X					X	X	X					

Discussion

- No employee works more than 6 consecutive days; the schedule produces a six-day work-stretch for one employee each week.
- This cannot be avoided since the solution is unique.

Alg 12.2.1 (Pinedo) overview

- Calculate *W*
- Step 1: Assign weekends off
- Step 2: Construct off-day pairs

Repeat for cycle of weeks:

- Step 3: Categorize week *i* workers
- Step 4: Assign off-day pairs for week *i*

Algorithm properties

- If all off-pairs are distinct ⇒ maximum work-stretch is 5 days.
- Schedule always satisfies the constraints.
- There exists an optimal cyclic schedule, that **may** be found by the algorithm.

Shift scheduling

- Fixed cycle
  - Month, week, day
- Predefined set of shift patterns
  - Each worker is assigned to exactly one pattern
  - Each pattern has its own cost
- **Assign workers to patterns such that staffing requirements are met and cost is minimized**

Shift scheduling

- A cycle consisting of m periods is given
- During period i, b_i employees are required
- There are n shift patterns, Shift pattern j is defined as $(a_{1j}, a_{2j}, \ldots, a_{mj})$. a_{ij} can be 0 or 1.
- Each employee is assigned to exactly one shift pattern
- Cost of assigning a person to shift j is c_j
- Decision variable x_j is the number of people assigned to shift j

Integer Programming problem

minimize $c_1x_1 + c_2x_2 + \ldots + c_nx_n$

subject to

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &\geq b_1 \\
a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &\geq b_2 \\
&\vdots \\
a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n &\geq b_m \\
x_j &\geq 0
\end{aligned}
$$

Integer Linear Program

- Matrix formulation:

min	$\sum_{j=1}^{n} c_j x_j$	min	$c\boldsymbol{x}$
s.t.	$\sum_{i=1}^{n} a_{ij} x_j \geq b_i$		$\boldsymbol{Ax} \geq \boldsymbol{b}$
	$x_j \geq 0$		$\boldsymbol{x} \geq 0$

Solution

- Strongly NP-hard in general
- Special structure in shift pattern matrix
  - e.g. having no split shifts
- Solve LP relaxation
  - Solution always integer when each column contains a contiguous set of ones:
    - LP optimization in polynomial time!

$$
A = \begin{bmatrix}
1 & 1 & 0 & 0 & 1 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 0 & 1
\end{bmatrix}
$$

Example 12.3.1 (Pinedo)

- Shift scheduling in a retail store

Pattern	Hours	Total Hours	Cost (€)
1	10AM – 6PM	8	50
2	1PM – 9PM	8	60
3	12PM – 6PM	6	30
4	10AM – 1PM	3	15
5	6PM – 9PM	3	16

Hour	Staff req.
10AM – 11AM	3
11AM – 12PM	4
12PM – 1PM	6
1PM – 2PM	4
2PM – 3PM	7
3PM – 4PM	8
4PM – 5PM	7
5PM – 6PM	6
6PM – 7PM	4
7PM – 8PM	7
8PM – 9PM	8

LP formulation

- c = (50, 60, 30, 15, 16)

$$\mathbf{A} = \begin{bmatrix} 1&0&0&1&0\\ 1&0&0&1&0\\ 1&0&1&1&0\\ 1&1&1&0&0\\ 1&1&1&0&0\\ 1&1&1&0&0\\ 1&1&1&0&0\\ 1&1&1&0&0\\ 0&1&0&0&1\\ 0&1&0&0&1\\ 0&1&0&0&1 \end{bmatrix} \qquad b = \begin{bmatrix} 3\\4\\6\\4\\7\\8\\7\\6\\4\\7\\8 \end{bmatrix}$$

Solution

- Linear program relaxation (to be given in **Optimization and Decision**):
- $\boldsymbol{x}$ = (0, 0, 8, 4, 8)

- A special case of shift scheduling, the **cyclic staffing problem**, is discussed in the following.
- Cyclic staffing is solvable in polynomial time.

Cyclic staffing

- An m-period cyclic schedule (e.g. 24 hours a day)
- Each period i has requirement b_i
- Each person works k consecutive periods and is free for the next $m - k$
- Again, c_j is the cost of putting a worker on pattern j
- Find minimum cost schedule

➢ **Example**: (5, 7)-cyclic staffing problem

(5,7)-cyclic staffing problem

- Integer Program with 7 different shift patterns:

$$A = \begin{bmatrix} 1&0&0&1&1&1&1\\ 1&1&0&0&1&1&1\\ 1&1&1&0&0&1&1\\ 1&1&1&1&0&0&1\\ 1&1&1&1&1&0&0\\ 0&1&1&1&1&1&0\\ 0&0&1&1&1&1&1 \end{bmatrix}$$

- Columns do not always have a contiguous set of 1's.

Solution

- Special structure makes the problem relatively simple.

Algorithm

- **Step 1**: Solve LP relaxation to obtain $x_1', x_2', \ldots, x_n'$ if integer STOP; otherwise continue
- **Step 2**: Formulate two new LPs adding the constraints:

$$x_1 + x_2 + \ldots + x_n = \lfloor x_1' + x_2' + \ldots + x_n' \rfloor$$
$$x_1 + x_2 + \ldots + x_n = \lceil x_1' + x_2' + \ldots + x_n' \rceil$$

- The best integer solution is optimal

Example

- (3,5)-cyclic staffing problem

$$A = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix} \qquad b = \begin{bmatrix} 3 \\ 4 \\ 6 \\ 4 \\ 7 \end{bmatrix}$$

Solution

- **Step 1**: Solve the LP relaxation

$$\min_x cx$$

subject to

$$Ax \le b$$

$$x \ge 0$$

- Solution: $\bar{x}' = (1.5, 0, 4.5, 0, 2.5)$

Non-integer

Solution

- Add together: $1.5 + 0 + 4.5 + 0 + 2.5 = 8.5$
- **Step 2a**: Add constraint:

$$x_1 + x_2 + x_3 + x_4 + x_5 = 8$$

- *No feasible solution*

- **Step 2b**: Add constraint:

$$x_1 + x_2 + x_3 + x_4 + x_5 = 9$$

- **Solution:**

$$\bar{x} = (2,\ 0,\ 4,\ 1,\ 2)$$

Optimal

Cyclic staffing: extensions

Possible **applications and extensions** of cyclic staffing:

1. Days-off scheduling
2. Cyclic staffing with overtime
3. Cyclic staffing with linear penalties for understaffing or overstaffing

1. Days-off scheduling

- Days-off scheduling problem can be represented as a cyclic staffing problem
  - all the shift patterns must be determined

- **Difficulty**: unknown cycle length
- **Difficulty**: many patterns $\Rightarrow$ larger problem

Example

- 2 days off in a week and maximum work-stretch of 6:

$$A = \left[\begin{array}{cccccc} 0 & 0 & 0 & \cdots & 1 & \cdots \\ 0 & 0 & 0 & \cdots & 1 & \cdots \\ 1 & 1 & 1 & \cdots & 1 & \cdots \\ 1 & 1 & 1 & \cdots & 1 & \cdots \\ 1 & 1 & 1 & \cdots & 1 & \cdots \\ 1 & 1 & 1 & \cdots & 0 & \cdots \\ 1 & 1 & 1 & \cdots & 0 & \cdots \\ \hdashline 1 & 0 & 1 & \cdots & 1 & \cdots \\ 0 & 1 & 0 & \cdots & 1 & \cdots \\ 1 & 1 & 1 & \cdots & 1 & \cdots \\ 1 & 1 & 1 & \cdots & 1 & \cdots \\ 1 & 1 & 1 & \cdots & 0 & \cdots \\ 1 & 1 & 0 & \cdots & 0 & \cdots \\ 0 & 0 & 1 & \cdots & 1 & \cdots \end{array}\right]$$

2. Cyclic staffing with overtime

- 24-hour operation (e.g. hospitals)
- 8-hour shifts with up to 8 hour overtime
- 3 shift of 8 hours each.

$$\mathbf{A} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0 \backslash 1} \\ \mathbf{0 \backslash 1} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0 \backslash 1} & \mathbf{1} \end{bmatrix} \qquad [\mathbf{0 \backslash 1}] = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

3. Penalties for under/overstaffing

- Demand not fixed
- Linear penalty $c_i^{'}$ for understaffing
- Linear penalty $c_i^{''}$ for overstaffing
- Let $x_i^{'}$ denote the level of understaffing
- **Integer Program** (*solved cyclic staffing algorithm*):

$$\min \quad \bar{c}\,\bar{x} + \bar{c}^{'}\bar{x}^{'} + \bar{c}^{''}\left(\bar{b} - A\bar{x} - \bar{x}^{'}\right)$$

$$\text{subject to} \quad A\bar{x} + I\bar{x}^{'} \geq \bar{b}$$

$$\bar{x}, \bar{x}^{'} \geq 0$$

Crew scheduling

- Given a set of jobs, as e.g. flight legs
- Crew requirements for each flight
- Find an assignment of crews to flight legs so that all crews return home (round trips).
- Flight legs have timings
  - Crews need to be in the right place at the right time
- Crews have work regulations
  - Break periods, maximum time without rest, etc.

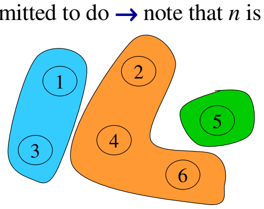
Crew scheduling

- Crews for all flights within an airline need to be coordinated.

- *Very important in transportation industry, especially in **airline industry***
  - planes, trains, trucks, buses, …
- Often it is a combined routing and crew scheduling problem: **vehicle routing** plus **crew scheduling**!

Problem definition

- Set of *m* jobs, say flight legs; each job has a start and end place and a time interval
- Set of *n* feasible combination of jobs a crew is permitted to do → note that *n* is very large!



$m = 6$

Set partitioning problem

Problem definition

- Cost c_j of round trip *j*
- Definitions:

$$a_{ij} = \begin{cases} 1 & \text{if leg } i \text{ is part of round trip } j \\ 0 & \text{otherwise} \end{cases}$$

$$x_j = \begin{cases} 1 & \text{if round trip } j \text{ is selected} \\ 0 & \text{otherwise} \end{cases}$$

Integer Program

$$\text{minimize} \quad c_1x_1 + c_2x_2 + \ldots + c_nx_n$$

$$\begin{aligned}\text{subject to} \quad a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= 1\\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= 1\\ &\vdots\\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n &= 1\\ x_j &\in \{0,1\}\end{aligned}$$

Each column is a round trip

Each row is a flight leg

Set Partitioning

- Optimization problem: find the set of round trips with minimum cost such that satisfy the constraints.
- **Set Partitioning problem**
- Constraints are called partitioning equations
- Variables x_j equal to 1 in a solution are a *partition*.

$$J^l = \{ j : x_j^l = 1 \}$$

- Problem is NP-hard
- Well studied like TSP, graph coloring, bin packing, etc.

Possible solution

- Use the concept of **row prices**.
- The vector $\bar{\rho}^l = (\rho_1^l, \rho_2^l, \ldots, \rho_m^l)$

  is a set of feasible row prices if

$$\sum_{i=1}^{m} \rho_i^l a_{ij} = c_j, \quad j \in J^l$$

- Price ρ_i^l is an estimate of the cost of covering a flight leg i using solution J^l

Change partition

- Let Z^1 (Z^2) denote the objective value of partition 1 (2)
- Then $Z^2 = Z^1 - \sum_{j \in J^2} \sum_{i=1}^{m} \left(\rho_i^1 a_{ij} - c_j \right)$
- *Potential savings* of including column j with respect to 1st partition is:

$$\sigma_j = \sum_{i=1}^{m} \left(\rho_i^1 a_{ij} - c_j \right)$$

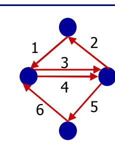
- If all negative then solution J^1 is optimal

Heuristic

- Start with some initial partition
- Construct a new partition as follows:
  - Find the column with highest potential savings
  - Include this column in new partition
  - If all jobs covered stop; otherwise repeat

A simple example

- What is the minimum cost set of round trips?
- Construct the constraint matrix



$$\sum_{j=1}^{n} a_{ij} x_j = 1 \quad i = 1, \ldots, m$$

Round trip	Legs	Cost
1	3,2,1	10
2	4,2,1	5
3	3,5,6	10
4	4,5,6	10

Heuristic (Algorithm 12.6.1)

- Initialization: Pick any solution, J^1
- **Step 1**: $J^2 = \{\}$
- **Step 2**: Find k, the round trip with maximum potential savings
- **Step 3**: Remove legs in k from other trips
- **Step 4**: Add k to J^2, remove trips with no legs
- **Step 5**: If no columns are candidates to include in J^2 STOP; otherwise go to Step 2.

Solving example

Round trip	Legs	Cost
1	3,2,1	10
2	4,2,1	5
3	3,5,6	10
4	4,5,6	10

- Let $J^1 = \{1,4\}$, cost = 20
- Find the row costs:

$$\bar{\rho}^1 = (\rho_1^1, \rho_2^1, \rho_3^1, \rho_4^1, \rho_5^1, \rho_6^1)$$

such that $\sum_{i=1}^{m} \rho_i^1 a_{ij} = c_j \quad j \in J^1$

Solving example

$$\sum_{i=1}^{m} \rho_i^1 a_{ij} = c_j \quad j \in J^1$$

Round trip	Legs	Cost
1	3,2,1	10
2	4,2,1	5
3	3,5,6	10
4	4,5,6	10

$j = 1$
$$\rho_1^1 a_{11} + \rho_2^1 a_{21} + \rho_3^1 a_{31} + \rho_4^1 a_{41} + \rho_5^1 a_{51} + \rho_6^1 a_{61} = c_1$$
$$\rho_1^1(1) + \rho_2^1(1) + \rho_3^1(1) + \rho_4^1(0) + \rho_5^1(0) + \rho_6^1(0) = 10$$
$$\rho_1^1 + \rho_2^1 + \rho_3^1 = 10$$

Solving example

$$\sum_{i=1}^{m} \rho_i^1 a_{ij} = c_j \quad j \in J^1$$

Round trip	Legs	Cost
1	3,2,1	10
2	4,2,1	5
3	3,5,6	10
4	4,5,6	10

$j = 4$
$$\rho_1^1 a_{14} + \rho_2^1 a_{24} + \rho_3^1 a_{34} + \rho_4^1 a_{44} + \rho_5^1 a_{54} + \rho_6^1 a_{64} = c_4$$
$$\rho_4^1 + \rho_5^1 + \rho_6^1 = 10$$

- We can consider the average: $\boxed{\rho_i^1 = 3.33}$

Algorithm 12.6.1

- Initialization: Pick any solution, J^1
- **Step 1**: $J^2 = \{\}$
- **Step 2**: Find k, the round trip with maximum potential savings
- **Step 3**: Remove legs in k from other trips
- **Step 4**: Add k to J^2, remove trips with no legs
- **Step 5**: If no columns are candidates to include in J^2 STOP; otherwise go to Step 2.

Potential savings

The cost of round trip j based on current prices

The real cost of round trip j

$$\sigma_j = \boxed{\sum_{i=1}^{m} \rho_i^1 a_{ij}} - c_j$$

Round trip	Legs	Cost	Potential savings
1	3,2,1	10	0
2	4,2,1	5	5
3	3,5,6	10	0
4	4,5,6	10	0

Algorithm 12.6.1

- Initialization: Pick any solution, J^1
- **Step 1**: $J^2 = \{\ \}$
- **Step 2**: Find k, the round trip with maximum potential savings
- **Step 3**: Remove legs in k from other trips
- **Step 4**: Add k to J^2, remove trips with no legs
- **Step 5**: If no columns are candidates to include in J^2 STOP; otherwise go to Step 2.

Solving example

Round trip	Legs	Cost
1	3,2,1	10
2	4,2,1	5
3	3,5,6	10
4	4,5,6	10

$J^2 = \{\ \}$

Round trip	Legs	Cost
1	3	10
2	4,2,1	5
3	3,5,6	10
4	5,6	10

$J^2 = \{2\}$

Back to Step 2: potential savings

$$\sigma_j = \sum_{i=1}^{m} \rho_i^1 a_{ij} - c_j$$

Round trip	Legs	Cost	Potential Savings
1	3	10	-6.6667
2	4,2,1	5	-
3	3,5,6	10	0
4	5,6	10	-3.333

$J^2 = \{2,3\}$
Cost = 15

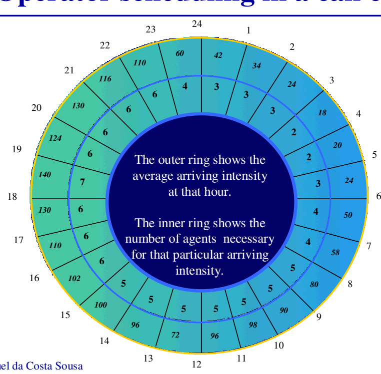
Back to Step 1

Round trip	Legs	Cost
1	3,2,1	10
2	4,2,1	5
3	3,5,6	10
4	4,5,6	10

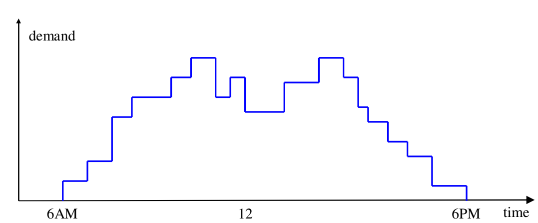
- Calculate row costs
- Calculate potential savings

$J^2 = \{2,3\}$
Cost = 15
$J^3 = \{\ \}$

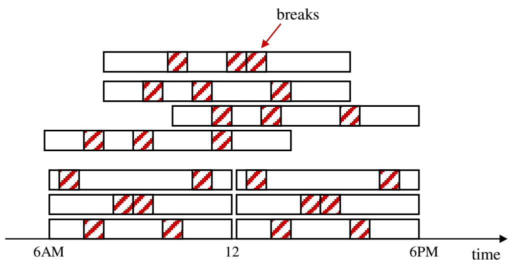
Operator scheduling in a call center



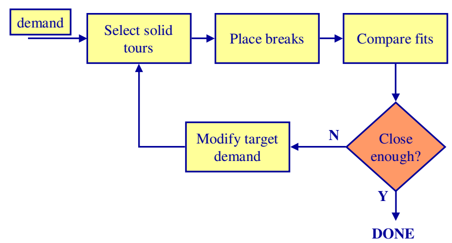
Call center scheduling: demand



Call centre scheduling: shift patterns



Solution framework (Fig 12.4)



Call centre scheduling

- Assign people to shifts to meet the demand and minimize costs
- It can be more complex: workers with different skills.
  - {English}, {English, French}, {French}

Call center scheduling: demand